下载APP

03 | 高性能IO模型：为什么单线程Redis能那么快？

2020-08-10 蒋德钧

Redis核心技术与实战

进入课程 >

讲述：蒋德钧
时长 13:22　大小 12.26M

你好，我是蒋德钧。

今天，我们来探讨一个很多人都很关心的问题：“为什么单线程的 Redis 能那么快？”

首先，我要和你厘清一个事实，我们通常说，Redis 是单线程，主要是指 **Redis 的网络 IO 和键值对读写是由一个线程来完成的，这也是 Redis 对外提供键值存储服务的主要流程。**但 Redis 的其他功能，比如持久化、异步删除、集群数据同步等，其实是由额外的线程执行的。

所以，严格来说，Redis 并不是单线程，但是我们一般把 Redis 称为单线程高性能，这样显得“酷”些。接下来，我也会把 Redis 称为单线程模式。而且，这也会促使你紧接着提问：“为什么用单线程？为什么单线程能这么快？”

要弄明白这个问题，我们就要深入地学习下 Redis 的单线程设计机制以及多路复用机制。之后你在调优 Redis 性能时，也能更有针对性地避免会导致 Redis 单线程阻塞的操作，例如执行复杂度高的命令。

好了，话不多说，接下来，我们就先来学习下 Redis 采用单线程的原因。

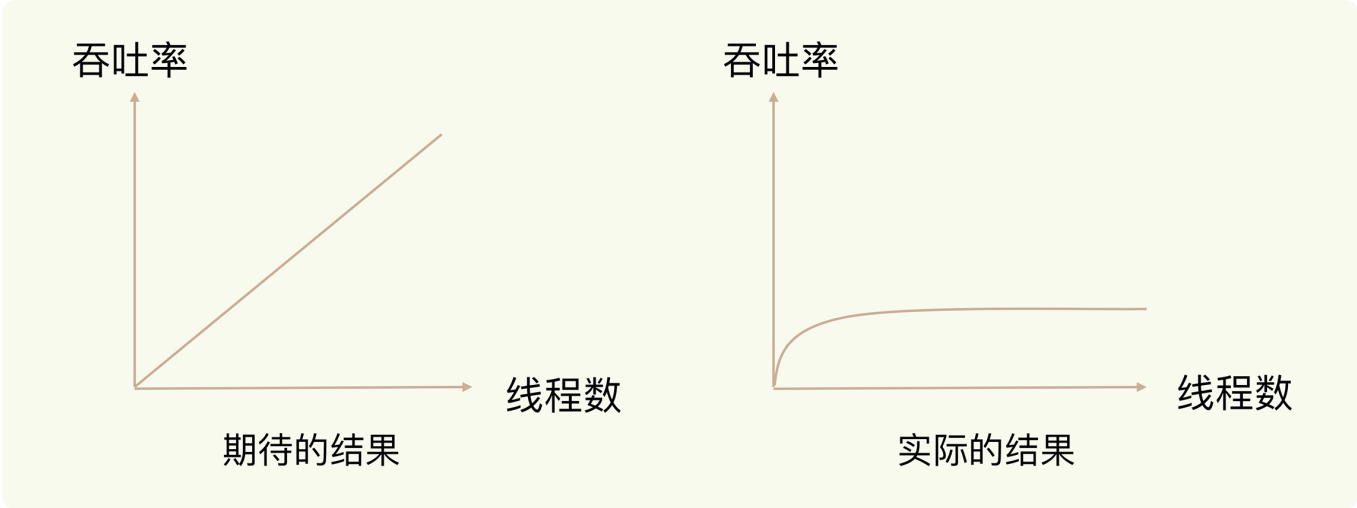
Redis 为什么用单线程？

要更好地理解 Redis 为什么用单线程，我们就要先了解多线程的开销。

多线程的开销

日常写程序时，我们经常会听到一种说法：“使用多线程，可以增加系统吞吐率，或是可以增加系统扩展性。”的确，对于一个多线程的系统来说，在有合理的资源分配的情况下，可以增加系统中处理请求操作的资源实体，进而提升系统能够同时处理的请求数，即吞吐率。下面的左图是我们采用多线程时所期待的结果。

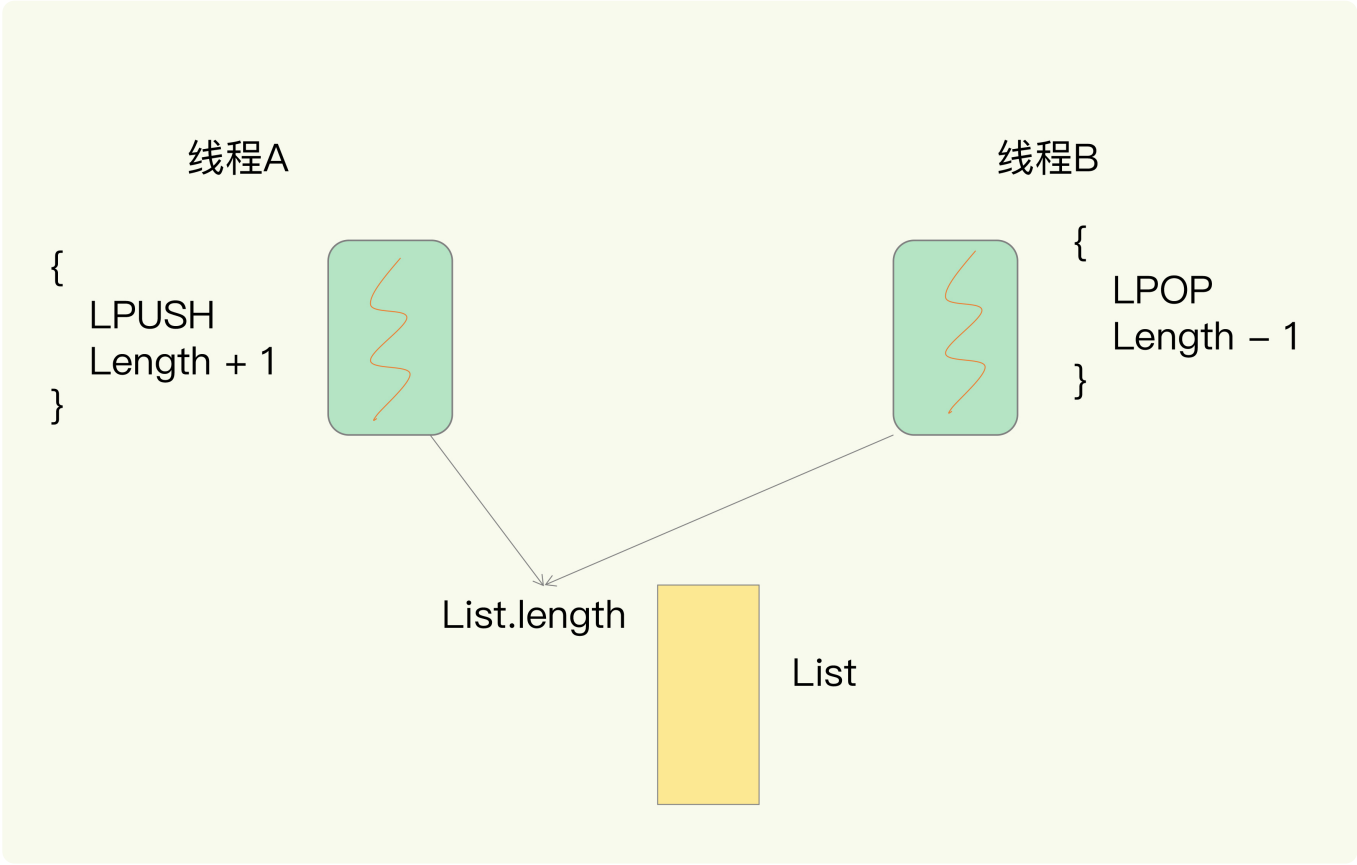
但是，请你注意，通常情况下，在我们采用多线程后，如果没有良好的系统设计，实际得到的结果，其实是右图所展示的那样。我们刚开始增加线程数时，系统吞吐率会增加，但是，再进一步增加线程时，系统吞吐率就增长迟缓了，有时甚至还会出现下降的情况。



线程数与系统吞吐率

为什么会出现这种情况呢？一个关键的瓶颈在于，系统中通常会存在被多线程同时访问的共享资源，比如一个共享的数据结构。当有多个线程要修改这个共享资源时，为了保证共享资源的正确性，就需要有额外的机制进行保证，而这个额外的机制，就会带来额外的开销。

拿 Redis 来说，在上节课中，我提到过，Redis 有 List 的数据类型，并提供出队（LPOP）和入队（LPUSH）操作。假设 Redis 采用多线程设计，如下图所示，现在有两个线程 A 和 B，线程 A 对一个 List 做 LPUSH 操作，并对队列长度加 1。同时，线程 B 对该 List 执行 LPOP 操作，并对队列长度减 1。为了保证队列长度的正确性，Redis 需要让线程 A 和 B 的 LPUSH 和 LPOP 串行执行，这样一来，Redis 可以无误地记录它们对 List 长度的修改。否则，我们可能就会得到错误的长度结果。这就是**多线程编程模式面临的共享资源的并发访问控制问题**。



多线程并发访问Redis

并发访问控制一直是多线程开发中的一个难点问题，如果没有精细的设计，比如说，只是简单地采用一个粗粒度互斥锁，就会出现不理想的结果：即使增加了线程，大部分线程也在等待获取访问共享资源的互斥锁，并行变串行，系统吞吐率并没有随着线程的增加而增加。

而且，采用多线程开发一般会引入同步原语来保护共享资源的并发访问，这也会降低系统代码的易调试性和可维护性。为了避免这些问题，Redis 直接采用了单线程模式。

讲到这里，你应该已经明白了“Redis 为什么用单线程”，那么，接下来，我们就来看看，为什么单线程 Redis 能获得高性能。

单线程 Redis 为什么那么快？

通常来说，单线程的处理能力要比多线程差很多，但是 Redis 却能使用单线程模型达到每秒数十万级别的处理能力，这是为什么呢？其实，这是 Redis 多方面设计选择的一个综合结果。

一方面，Redis 的大部分操作在内存上完成，再加上它采用了高效的数据结构，例如哈希表和跳表，这是它实现高性能的一个重要原因。另一方面，就是 Redis 采用了**多路复用机制**，使其在网络 IO 操作中能并发处理大量的客户端请求，实现高吞吐率。接下来，我们就重点学习下多路复用机制。

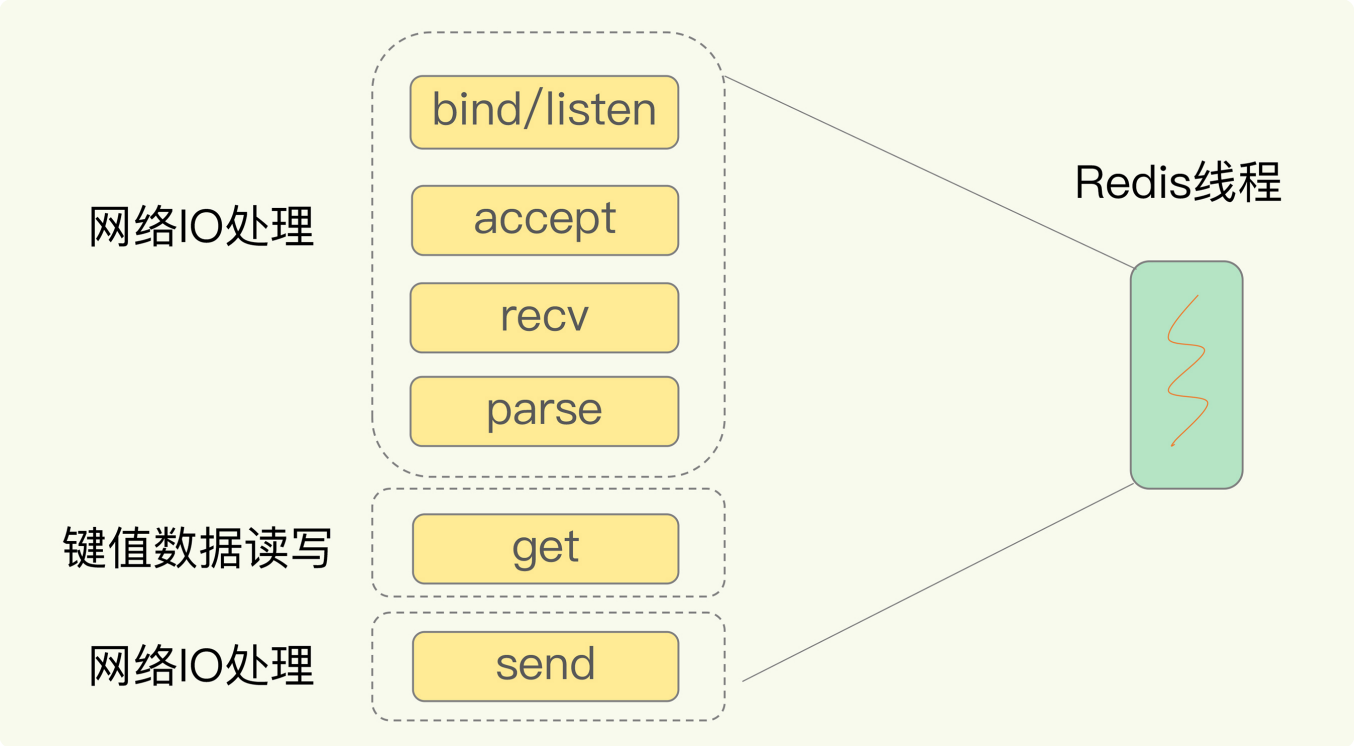
首先，我们要弄明白网络操作的基本 IO 模型和潜在的阻塞点。毕竟，Redis 采用单线程进行 IO，如果线程被阻塞了，就无法进行多路复用了。

基本 IO 模型与阻塞点

你还记得我在第一节课介绍的具有网络框架的 SimpleKV 吗？

以 Get 请求为例，SimpleKV 为了处理一个 Get 请求，需要监听客户端请求（bind/listen），和客户端建立连接（accept），从 socket 中读取请求（recv），解析客户端发送请求（parse），根据请求类型读取键值数据（get），最后给客户端返回结果，即向 socket 中写回数据（send）。

下图显示了这一过程，其中，bind/listen、accept、recv、parse 和 send 属于网络 IO 处理，而 get 属于键值数据操作。既然 Redis 是单线程，那么，最基本的一种实现是在一个线程中依次执行上面说的这些操作。



Redis基本IO模型

但是，在这里的网络 IO 操作中，有潜在的阻塞点，分别是 accept() 和 recv()。当 Redis 监听到一个客户端有连接请求，但一直未能成功建立起连接时，会阻塞在 accept() 函数这里，导致其他客户端无法和 Redis 建立连接。类似的，当 Redis 通过 recv() 从一个客户端读取数据时，如果数据一直没有到达，Redis 也会一直阻塞在 recv()。

这就导致 Redis 整个线程阻塞，无法处理其他客户端请求，效率很低。不过，幸运的是，socket 网络模型本身支持非阻塞模式。

非阻塞模式

Socket 网络模型的非阻塞模式设置，主要体现在三个关键的函数调用上，如果想要使用 socket 非阻塞模式，就必须要了解这三个函数的调用返回类型和设置模式。接下来，我们就重点学习下它们。

在 socket 模型中，不同操作调用后会返回不同的套接字类型。socket() 方法会返回主动套接字，然后调用 listen() 方法，将主动套接字转化为监听套接字，此时，可以监听来自客户端的连接请求。最后，调用 accept() 方法接收到达的客户端连接，并返回已连接套接字。

调用方法	返回套接字类型	非阻塞模式	效果
socket()	主动套接字		
listen()	监听套接字	可设置	accept()非阻塞
accept()	已连接套接字	可设置	send()/recv()非阻塞

Redis套接字类型与非阻塞设置

针对监听套接字，我们可以设置非阻塞模式：当 Redis 调用 accept() 但一直未有连接请求到达时，Redis 线程可以返回处理其他操作，而不用一直等待。但是，你要注意的是，调用 accept() 时，已经存在监听套接字了。

虽然 Redis 线程可以不用继续等待，但是总得有机制继续在监听套接字上等待后续连接请求，并在有请求时通知 Redis。

类似的，我们也可以针对已连接套接字设置非阻塞模式：Redis 调用 recv() 后，如果已连接套接字上一直没有数据到达，Redis 线程同样可以返回处理其他操作。我们也需要有机制继续监听该已连接套接字，并在有数据达到时通知 Redis。

这样才能保证 Redis 线程，既不会像基本 IO 模型中一直在阻塞点等待，也不会导致 Redis 无法处理实际到达的连接请求或数据。

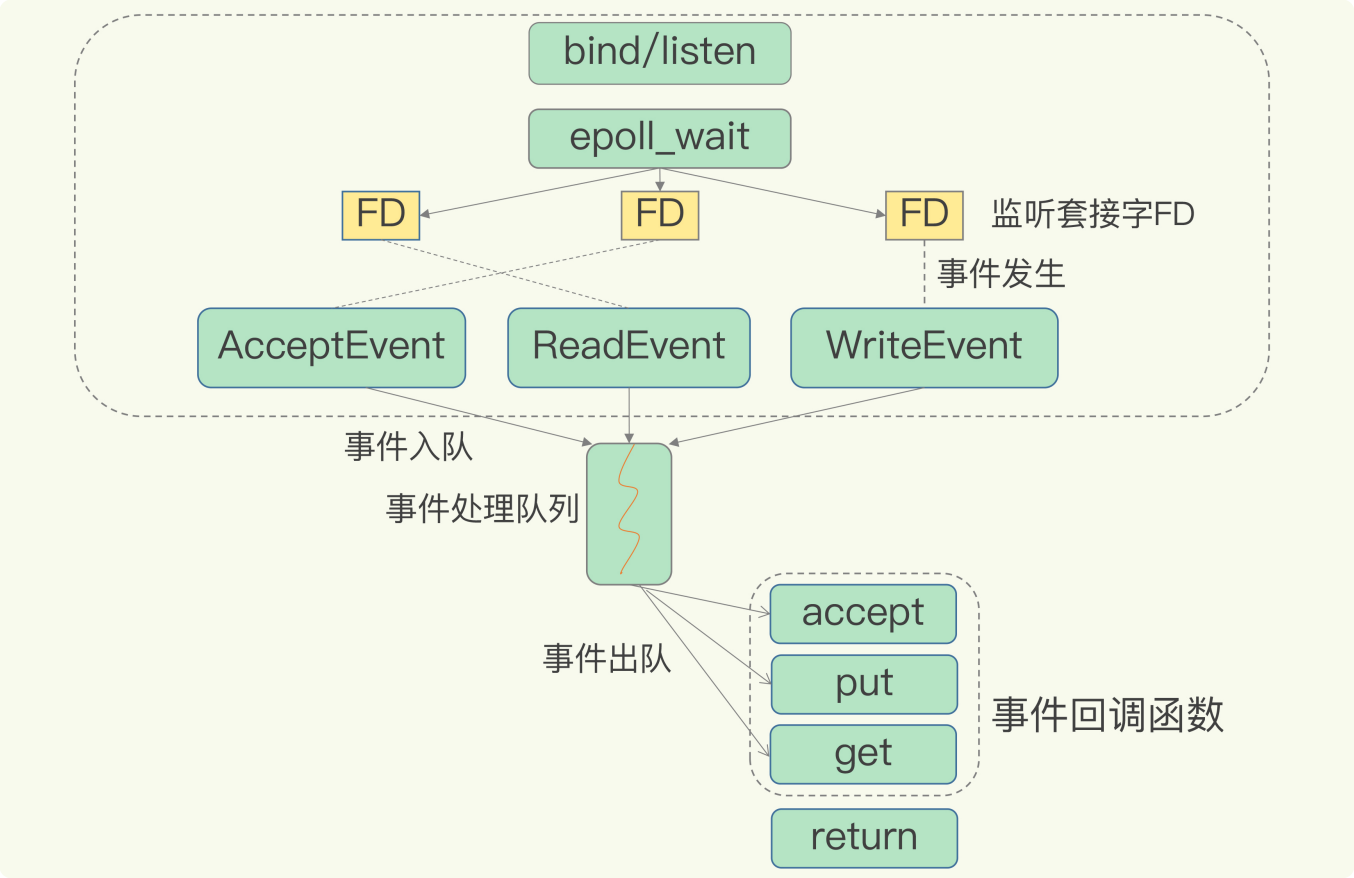
到此，Linux 中的 IO 多路复用机制就要登场了。

基于多路复用的高性能 I/O 模型

Linux 中的 IO 多路复用机制是指一个线程处理多个 IO 流，就是我们经常听到的 select/epoll 机制。简单来说，在 Redis 只运行单线程的情况下，**该机制允许内核中，同时存在多个监听套接字和已连接套接字**。内核会一直监听这些套接字上的连接请求或数据请求。一旦有请求到达，就会交给 Redis 线程处理，这就实现了一个 Redis 线程处理多个 IO 流的效果。

下图就是基于多路复用的 Redis IO 模型。图中的多个 FD 就是刚才所说的多个套接字。Redis 网络框架调用 epoll 机制，让内核监听这些套接字。此时，Redis 线程不会阻塞在某

一个特定的监听或已连接套接字上，也就是说，不会阻塞在某一个特定的客户端请求处理上。正因为此，Redis 可以同时和多个客户端连接并处理请求，从而提升并发性。



基于多路复用的Redis高性能IO模型

为了在请求到达时能通知到 Redis 线程，select/epoll 提供了**基于事件的回调机制**，即**针对不同事件的发生，调用相应的处理函数**。

那么，回调机制是怎么工作的呢？其实，select/epoll 一旦监测到 FD 上有请求到达时，就会触发相应的事件。

这些事件会被放进一个事件队列，Redis 单线程对该事件队列不断进行处理。这样一来，Redis 无需一直轮询是否有请求实际发生，这就可以避免造成 CPU 资源浪费。同时，Redis 在对事件队列中的事件进行处理时，会调用相应的处理函数，这就实现了基于事件的回调。因为 Redis 一直在对事件队列进行处理，所以能及时响应客户端请求，提升 Redis 的响应性能。

为了方便你理解，我再以连接请求和读数据请求为例，具体解释一下。

这两个请求分别对应 Accept 事件和 Read 事件，Redis 分别对这两个事件注册 accept 和 get 回调函数。当 Linux 内核监听到有连接请求或读数据请求时，就会触发 Accept 事件和 Read 事件，此时，内核就会回调 Redis 相应的 accept 和 get 函数进行处理。

这就像病人去医院瞧病。在医生实际诊断前，每个病人（等同于请求）都需要先分诊、测体温、登记等。如果这些工作都由医生来完成，医生的工作效率就会很低。所以，医院都设置了分诊台，分诊台会一直处理这些诊断前的工作（类似于 Linux 内核监听请求），然后再转交给医生做实际诊断。这样即使一个医生（相当于 Redis 单线程），效率也能提升。

不过，需要注意的是，即使你的应用场景中部署了不同的操作系统，多路复用机制也是适用的。因为这个机制的实现有很多种，既有基于 Linux 系统下的 select 和 epoll 实现，也有基于 FreeBSD 的 kqueue 实现，以及基于 Solaris 的 evport 实现，这样，你可以根据 Redis 实际运行的操作系统，选择相应的多路复用实现。

小结

今天，我们重点学习了 Redis 线程的三个问题：“Redis 真的只有单线程吗？” “为什么用单线程？” “单线程为什么这么快？”

现在，我们知道了，Redis 单线程是指它对网络 IO 和数据读写的操作采用了一个线程，而采用单线程的一个核心原因是避免多线程开发的并发控制问题。单线程的 Redis 也能获得高性能，跟多路复用的 IO 模型密切相关，因为这避免了 accept() 和 send()/recv() 潜在的网络 IO 操作阻塞点。

搞懂了这些，你就走在了很多人的前面。如果你身边还有不清楚这几个问题的朋友，欢迎你分享给他 / 她，解决他们的困惑。

另外，我也剧透下，可能你也注意到了，2020 年 5 月，Redis 6.0 的稳定版发布了，Redis 6.0 中提出了多线程模型。那么，这个多线程模型和这节课所说的 IO 模型有什么关联？会引入复杂的并发控制问题吗？会给 Redis 6.0 带来多大提升？关于这些问题，我会在后面的课程中和你具体介绍。

每课一问

这节课，我给你提个小问题，在“Redis 基本 IO 模型”图中，你觉得还有哪些潜在的性能瓶颈吗？欢迎在留言区写下你的思考和答案，我们一起交流讨论。

34 人觉得很赞 | 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 数据结构：快速的Redis有哪些慢操作？

下一篇 04 | AOF日志：宕机了，Redis如何避免数据丢失？

精选留言 (38)

写留言

Kaito
2020-08-10

Redis单线程处理IO请求性能瓶颈主要包括2个方面：

1、任意一个请求在server中一旦发生耗时，都会影响整个server的性能，也就是说后面的请求都要等前面这个耗时请求处理完成，自己才能被处理到。耗时的操作包括以下几种：
a、操作bigkey：写入一个bigkey在分配内存时需要消耗更多的时间，同样，删除big...

展开

7 102

Darren
2020-08-10

1.big key的操作。
2.潜在的大量数据操作，比如 key *或者get all之类的操作，所以才引入了scan的相关操作。
3.特殊的场景，大量的客户端接入。
...

展开

4 21

每天晒白牙
2020-08-13

Redis 的单线程指 Redis 的网络 IO 和键值对读写由一个线程来完成的（这是 Redis 对外提供键值对存储服务的主要流程）
Redis 的持久化、异步删除、集群数据同步等功能是由其他线程而不是主线程来执行的，所以严格来说，Redis 并不是单线程
...

展开

5

一步
2020-08-11

Redis的事件处理队列只有一个吗？不同的事件的优先级都是一样的吗？只是简单的按照对接的先进先出的特性依次进行处理的吗？

展开

4

test
2020-08-10

单线程同步非阻塞读取网络IO的时候会有性能瓶颈，如果读取的内容过多的时候

3

注定非凡
2020-08-11

1，作者讲了什么？
redis实现单线程实现高性能IO的设计机制
2，作者是怎么把这事给讲明白的？
作者首先从简单的网络通信socket讲起，引出了非阻塞socket，由此谈到了著名的I/O多路复用，Linux内核的select/epoll机制...
展开

2

咸鱼
2020-08-10

这章让我对IO多路复用的理解又深了些
展开

2

Mr.蜜
2020-08-12

忘记提另一个问题，既然老师说道select和epoll，为什么不提一下poll呢？

2 1

曾轼麟
2020-08-10

虽然单线程很快，没有锁的单线程更快借助CPU的多级缓存可以把性能发挥到最大。但是随着访问量的增加，以及数据量的增加，IO的写入写出会成为性能瓶颈。10个socket的IO吞吐处理肯定比1000个socket吞吐处理的快，为了解决这个问题，Redis6引入了IO多线程的方式以及client缓冲区，在实际指令处理还是单线程模式。在IO上变成的了【主线程】带着众多【IO线程】进行IO，IO线程听从主线程的指挥是写入还是写出。Read的时...
展开

1

每天晒白牙
2020-08-10

理解的深入了
展开

2 1

范闲
2020-08-17

Redis的网络模式是单reactor模式。non-blocking io + epoll

smily
2020-08-17

如果把采用进程模型和I/O模型、epoll依次讲解，附带 操作系统刷盘机制就完美了

学习个球
2020-08-14

每日一答：
Redis 虽然利用了 I/O 多路复用技术，采用单线程来处理这些事件，避免了多线程的缺点。但是
1. 在客户端的并发量很大的情况下，单线程始终有它的瓶颈在。
2.还有一点就是事件入队的时候可能存在竞争关系导致性能问题吗？

展开

zhou
2020-08-14

老师分析的 redis io 模型中，redis 线程是循环处理每个事件的。如果其中一个事件比较耗时，会影响后面事件的及时处理。

展开

作者回复: 是的，所以如果有慢操作的话，就会影响其他操作了。

Mr.蜜
2020-08-12

首先看完课题，发现几个问题，Redis的持久化到底是多线程，还是子进程？copy on write又是怎么用在持久化中的？第二个，BIO和NIO看了半天才反应过来，难道他不叫Block IO和non Block IO吗？第三个算是回答，Redis6.0中的多线程是IO多线程，就是说Redis把网络消息的收发让其他线程去处理，这样无疑可以让Redis的工作线程只处理与数据有关的事务，而把数据的收发全让其他线程去处理，这样做就意味着Redis6.0的性能更好。

展开

gerry pang
2020-08-12

老师，我有一点没太明白，
‘’ Redis 的其他功能，比如持久化、异步删除、集群数据同步等，其实是由额外的线程

执行的。 ”
我看例如镜像文件的创建，redis不是后台fork出子进程吗？

展开

子龙
2020-08-11

1. 单线程是指多个连接或者请求使用同一个线程处理，类似浏览器多个用户的点击使用同一个线程处理
2. 单线程减少多线程引入的访问控制带来的开销，例如同步访问共享数据
3. IO 模型是OS的IO模型，Redis只是使用了这种比较友好的Epoll模型，由系统负责监听客户端的连接等，通知Redis工作线程，属于流程优化。

展开

humor
2020-08-11

当 Linux 内核监听到有连接请求或读数据请求时，就会触发 Accept 事件和 Read 事件，此时，内核就会回调 Redis 相应的 accept 和 get 函数进行处理。
这里的accept和get函数的调用是redis在应用侧调用的，而不是linux内核调用的吧，因为linux内核只是通知redjs有事件到达了，需要处理

展开

LindaWang
2020-08-11

网络I/O瓶颈：随着业务场景越来越复杂，需要更大的QPS，如果部署过多的Redis服务器就会导致管理的Redis服务器过多，维护代价大。采用多线程，可以充分利用多核分摊Redis同步I/O读写负荷

第四范式
2020-08-11

请教一个问题。“Redis 单线程对该事件队列不断进行处理。”这句话的意思是。对于某个事件的回调函数如果处理的时间较多。就会造成线程阻塞吧。疑问如下：(1)事件必须处理完某一个，下一个才 继续进行吧?(2) 线程来处理IO。不算阻塞。只有在处理IO之前，等待数据来的时间，才算阻塞？希望老师帮忙解答答一下。谢谢您。

展开